Lecture 11

Securing Web Applications

Recommended readings

- http://www.theserverside.com/tt/articles/content/TomcatSecurity/TomcatSecurity.pdf
- http://localhost:8080/tomcat-docs/security-manager-howto.html
- http://courses.coreservlets.com/Course-Materials/msajsp.html

Security

- Web applications contain resources that can be accessed by many users
- Internet is unprotected, open network
- Major issues:
  - Preventing unauthorized users from accessing sensitive data.
    - Access restriction
      - Identifying which resources need protection
      - Identifying who should have access to them
    - Authentication
      - Identifying users to determine if they are one of the authorized ones
  - Preventing attackers from stealing network data while it is in transit.
    - Encryption (usually with SSL)

Declarative Security

- None of the individual servlets or JSP pages need any security-aware code.
  - Instead, both of the major security aspects are handled by the server.
- To prevent unauthorized access
  - Use the Web application deployment descriptor (*web.xml*) to declare that certain URLs need protection.
  - Designate authentication method that server uses to identify users.
  - At request time, the server automatically prompts users for usernames and passwords when they try to access restricted resources, automatically checks the results against a server-specific set of usernames and passwords, and automatically keeps track of which users have previously been authenticated.
- To safeguard network data
  - Use the deployment descriptor to stipulate that certain URLs should be accessible only with SSL. If users try to use a regular HTTP connection to access one of these URLs, the server automatically redirects them to the HTTPS (SSL) equivalent.

Programmatic Security

- Protected servlets and JSP pages at least partially manage their own security.
  - Much more work, but totally portable.
    - No server-specific piece. Also no web.xml entries needed and a bit more flexibility is possible.
- To prevent unauthorized access
  - Each servlet or JSP page must either authenticate the user or verify that the user has been authenticated previously.
- To safeguard network data
  - Each servlet or JSP page has to check the network protocol used to access it.
  - If users try to use a regular HTTP connection to access one of these URLs, the servlet or JSP page must manually redirect them to the HTTPS (SSL) equivalent.

Form-Based Authentication

- When a not-yet-authenticated user tries to access a protected resource:
  - Server automatically redirects user to web page with an HTML form that asks for username and password
  - Username and password checked against database of usernames, passwords, and roles (user categories)
  - If login successful and role matches, page shown
  - If login unsuccesful, error page shown
  - If login successful but role does not match, 403 error given (but you can use error-page and error-code)
- When an already authenticated user tries to access a protected resource:
  - If role matches, page shown
  - If role does not match, 403 error given
  - Session tracking used to tell if user already authenticated

BASIC Authentication

- When a not-yet-authenticated user tries to access a protected resource:
  - Server sends a 401 status code to browser
  - Browser pops up dialog box asking for username and password, and they are sent with request in Authorization request header
  - Username and password checked against database of usernames, passwords, and roles (user categories)
  - If login successful and role matches, page shown
  - If login unsuccesful or role does not match, 401 again
- When an already authenticated user tries to access a protected resource:
  - If role matches, page shown
  - If role does not match, 401 error given
  - Request header used to tell if user already authenticated

Form-Based Authentication (Declarative Security)

1. Set up usernames, passwords, and roles
   - Designate a list of users and associated passwords and abstract role(s) such as normal user or administrator.
   - This is a completely server-specific process.
   - Simplest Tomcat approach: use *install_dir/conf/tomcat-users.xml*:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tomcat-users>
<user username="john" password="nhoj“ roles="registered-user" />
<user username="jane" password="enaj“ roles="registered-user" />
<user username="juan" password="nauj“ roles="administrator" />
<user username="juana" password="anauj“ roles="administrator,registered-user" />
</tomcat-users>
```

Form-Based Authentication (Declarative Security)

2. Tell server that you are using form-based authentication. Designate locations of login and login-failure page.
   – Use the *web.xml* login-config element with auth-method of FORM and form-login-config with locations of pages.

```
<web-app> …
<login-config>
   <auth-method>FORM</auth-method>
   <form-login-config>
      <form-login-page>/login.jsp</form-login-page>
      <form-error-page>/login-error.html</form-error-page>
   </form-login-config>
</login-config>
…</web-app>
```


Form-Based Authentication (Declarative Security)

3. Create a login page (HTML or JSP)
   – HTML form with ACTION of j_security_check, METHOD of POST, textfield named j_username, and password field named j_password.

```
<FORM ACTION="j_security_check" METHOD="POST">
…
<INPUT TYPE="TEXT" NAME="j_username">
…
<INPUT TYPE="PASSWORD" NAME="j_password">
…
</FORM>
```

   – For the username, you can use a list box, combo box, or set of radio buttons instead of a textfield.

Form-Based Authentication (Declarative Security)

4. Create page for failed login attempts.
   – No specific content is mandated.
   – Perhaps just “username and password not found” and give a link back to the login page.
   – This can be either an HTML or a JSP document.

Form-Based Authentication (Declarative Security)

5. Specify URLs to be password protected.
   – Use security-constraint element of *web.xml*. Two subelements: the first (web-resource-collection) designates URLs to which access should be restricted; the second (auth-constraint) specifies abstract roles that should have access to the given URLs. Using auth-constraint with no role-name means no *direct* access is allowed.

```
<web-app ...>…
<security-constraint>
   <web-resource-collection>
      <web-resource-name>Sensitive</web-resource-name>
      <url-pattern>/sensitive/*</url-pattern>
   </web-resource-collection>
   <auth-constraint>
      <role-name>administrator</role-name>
      <role-name>executive</role-name>
   </auth-constraint>
</security-constraint>
<login-config>...</login-config>…
</web-app>
```

Form-Based Authentication (Declarative Security)

6. List all possible abstract roles (types of users) that will be granted access to *any* resource
   – Required only with servlets 2.4, but even there, most servers do not enforce this

```
<web-app ...>
...
<security-role>
   <role-name>administrator</role-name>
   <role-name>executive</role-name>
</security-role>
</web-app>
```


Form-Based Authentication (Declarative Security)

7. Specify which URLs require SSL.
   – If server supports SSL, you can stipulate that certain resources are available only through encrypted HTTPS (SSL) connections. Use the user-data-constraint subelement of security-constraint. Only full J2EE servers are *required* to support SSL.

```
<web-app ...>
...
<security-constraint>
…
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
</web-app>
```


Form-Based Authentication (Declarative Security)

8. Turn off the invoker servlet.
   – You protect certain URLs that are associated with registered servlet or JSP names. The *http://host/prefix/servlet/Name* format of default servlet URLs will probably not match the pattern. Thus, the security restrictions are bypassed when the default URLs are used.
   – Disabling it
      • In each Web application, redirect requests to other servlet by normal *web.xml* method

```
<url-pattern>/servlet/*</urlpattern>
```


Form-Based vs. BASIC Authentication

- Advantages of form-based
  – Consistent look and feel
  – Fits model users expect from ecommerce sites
- Disadvantage of form-based
  – Can fail if server is using URL rewriting for session tracking. Can fail if browser has cookies disabled.
- Advantages of BASIC
  – Doesn't rely on session tracking
  – Easier when you are doing it yourself (programmatic)
- Disadvantage of BASIC
  – Small popup dialog box seems less familiar to most users
- Other auth-method options
  – CLIENT-CERT (X 509 certificates)
  – DIGEST (Not widely supported by browsers)

BASIC Authentication

1. Set up usernames, passwords, and roles.
   – Same as for form-based authentication. Server-specific.
2. Tell the server that you are using BASIC authentication. Designate the realm name.
   – Use the *web.xml* login-config element with an auth-method subelement of BASIC and a realm-name subelement (generally used as part of the title of the dialog box that the browser opens).

```
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>Some Name</realm-name>
</login-config>
```


BASIC Authentication

3. Specify which URLs should be password protected.
   – Same as with form-based authentication.
4. List all possible roles (categories of users) that will access any protected resource
   – Same as with form-based authentication
5. Specify which URLs should be available only with SSL.
   – Same as with form-based authentication.
6. Turn off the invoker servlet.
   – Same as with form-based authentication.

Problems with Pure Declarative Security

- Access is all-or-nothing
  - Users can access a resource or be denied access to it.
  - No changes depending on who accesses resource.
- Access based on exact password matches
  - Controlled by server.
- Involves server-specific component
  - Thus is not completely portable
    - Servers must support *some* way to designate users, passwords, and roles. But different servers do it different ways.
- All pages use same mechanism
  - Can't mix form-based and BASIC in same web app
- Requires web.xml entries

Combining Container-Managed and Programmatic Security

- Rely on the container (server) for usernames, passwords, and roles
  - Form-based or BASIC authentication as before
- Manage access explicitly from within the servlets or JSP pages
  - For example, you can change the result of a particular page depending on who accesses it. With pure declarative security, it is all or nothing.
- Use the following HttpServletRequest methods
  - isUserInRole
  - getRemoteUser
  - getUserPrincipal

Pure Programmatic Security

- Idea
  - Each protected resource authenticates users and decides what (if any) access to grant
- Advantages
  - Totally portable
    - No server-specific component
  - Permits custom password-matching strategies
  - No *web.xml* entries needed
- Disadvantages
  - Much harder to write and maintain
  - Each and every resource has to use the code
  - You can build reusable infrastructure (e.g., servlets that inherit from certain class or custom JSP tags), but it is still a lot of work.

Pure Programmatic Security

1. Check whether there is an Authorization request header.
   - If there is no such header, go to Step 5.
2. Get the encoded username/password string.
   - If there is an Authorization header, it should have the following form:
     - Authorization: Basic encodedData
   - Skip over the word Basic—the remaining part is the username and password represented in base64 encoding.
3. Reverse the base64 encoding of the username/password string.
   - Use the decodeBuffer method of the BASE64Decoder class. This method call results in a string of the form username:password. The BASE64Decoder class is bundled with the JDK; in JDK it can be found in the sun.misc package in *jdk_install_dir/jre/lib/rt.jar*.

Pure Programmatic Security

4. Check the username and password.
   - The most common approach is to use a database or a file to obtain the real usernames and passwords. For simple cases, it is also possible to place the password information directly in the servlet. If the incoming username and password match one of the reference username/password pairs, return the page. If not, go to Step 5. With this approach you can provide your own definition of “match.” With container-managed security, you cannot.
5. When authentication fails, send the appropriate response to the client.
   - Return a 401 (Unauthorized) response code and a header of the following form:
     - **WWW-Authenticate: BASIC realm="some-name"**
   - This response instructs the browser to pop up a dialog box telling the user to enter a name and password for some-name, then to reconnect with that username and password embedded in a single base64 string inside the Authorization header.

Using Programmatic Security with SSL

- Determining If SSL Is in Use
  - request.getScheme (returns "https" or "http")
  - request.isSecure (returns true or false)
- Redirecting Non-SSL Requests
  - response.sendRedirect
- Discovering the Number of Bits in the Key
  - request.getAttribute("javax.servlet.request.key_size")
- Looking Up the Encryption Algorithm
  - request.getAttribute("javax.servlet.request.cipher_suite")
- Accessing Client X509 Certificates
  - request.getAttribute("javax.servlet.request.X509Certificate")